B1 Problem Statement

Unit Pricing

Determine the best buy (the lowest per unit cost) between two items. The inputs will be the weight in ounces and the cost in dollars. Display whether the first or the second item is cheaper. Also display the cost per ounce for both items. At least two decimal places should be shown. The outputs should be shown with units.

Example

Weight 1: 30

Cost 1: 5.00

Weight 2: 50

Cost 2: 6.50

Answer: item 2 is a better buy. Item 1 is $0.166 per ounce. Item 2 is $0.13 per ounce.

B1 Solution
Cost per Ounce

```
int main()
{
	double W1, W2, P1, P2, CPU1, CPU2;

	std::cout << "\nenter weight 1: ";
	std::cin >> W1;
	std::cout << "\nenter price 1: ";
	std::cin >> P1;
	CPU1= P1/W1;
	std::cout << "\nenter weight 2: ";
	std::cin >> W2;
	std::cout << "\nenter price 2: ";
	std::cin >> P2;
	CPU2 = P2 / W2;
	if (CPU2 >= CPU1) {
		std::cout << "\nItem 1 is a better purchase or equal purchase. 1 is $"
		<< CPU1 << " per ounce.  2 is $" << CPU2 << " per ounce.";
		}
	else
	{
		std::cout << "\nItem 2 is a better purchase. 1 is $" << CPU1
			<< " per ounce.  2 is $" << CPU2 << " per ounce.";
	}
	return 0;
}
```

B2 Problem Statement
Dice Game

There are several games and puzzles that are done using dice. This problem will use standard dice that have six sides, numbered 1, 2, 3, 4, 5, and 6. If we roll some dice, the total sum is the sum of values on the top face of the dice. Given the total sum of three dice that were rolled, output all the combinations of rolls that are equal to the total sum. Your program should also output the total number of combinations found.

Input Format

The input format is the total as a whole number.

Output Format

All of the dice combinations should be outputted first, one per line. Finally, the total number of dice combinations should be outputted on its own line.

Sample Input

15

Sample Output

(3, 6, 6)
(4, 5, 6)
(4, 6, 5)
(5, 4, 6)
(5, 5, 5)
(5, 6, 4)
(6, 3, 6)
(6, 4, 5)
(6, 5, 4)
(6, 6, 3)
10

```
import itertools


def get_input():
    return int(input())


numbers = [1, 2, 3, 4, 5, 6]
num_dice = 3
results = []
total = get_input()
count = 0

for roll in itertools.product(numbers, repeat=num_dice):
    if sum(roll) == total:
        print("({})".format(",".join(str(n) for n in roll)))
        count += 1

print(count)
```

B3 Problem Statement

Decoding Roman Numerals

Roman numerals were invented by the ancient Romans and served as the Western number system for over a thousand years. The Roman system uses letters for specific values: “I” for one; “V” for five; “X” for ten; “L” for fifty; “C” for hundred; “D” for five hundred; “M” for thousand. The symbols were generally listed from highest value to lowest. E.g. MCCLXII represents 1262: M is 1000; CC is 200; LX is 60; and II is 2. The exceptions to the ordering are: “I” before a “V” represents 4; “I” before an “X” represents 9. Similarly, “XL” represents 40; “XC” 90; “CD” 400; “CM” 900. (based on Wikipedia’s “Roman numerals”).

Given a Roman Numeral of 8 chars or less, print the decimal equivalent.

Examples:

Input: XIV

Output: 14

Input: MCMLV

Output: 1955

Input: MMXVII

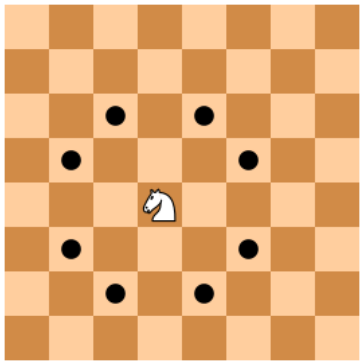
Output: 2017

B3 Solution
Decoding Roman Numerals

```
int main()
{
	char RC[10];
	int Fifty, Ten, Five, One;
	int n, i, sub;
	Fifty = 0; Ten = 0; Five = 0; One = 0; sub = 50;
	std::cout << "\nenter number of letters: ";
	std::cin >> n;
	for (i = 0; i < n; i++) {
		std::cin >> RC[i];
		std::cout << "\n " << RC[i] ;
	}

	for (i = 0; i < n; i++) {
		if (RC[i] == 'L') { Fifty++; }
		if (RC[i] == 'X') {
			Ten++;
			if (i < n - 1 && RC[i + 1] == 'L')
			{
				Ten = -Ten;
			}
		}
		if (RC[i] == 'V') {  Five++;}
		if (RC[i] == 'I') {
			One++;
			if (i < n && (RC[i+1] == 'X' || RC[i+1] == 'V'))
			{
				One = -One;
			}
		}
		std::cout << "\nValue is " << 50 * Fifty + 10 * Ten + 5 * Five + One;
		}

	std::cout << "\nquit? enter q: ";
	std::cin >> quit;

	return 0;
}
```

B4 Problem Statement
Chess

Chess is a very popular board game played between two players on a 8 x 8 checkered board. There are several different pieces used in the game. Given its initial location, calculate all the valid single moves a **knight** chess piece can make on the board. You can assume no other pieces on the board except the one that is given. Moves cannot go outside the bounds of the board. See below for descriptions of how the knight piece can move. Each square on the board is numbered, starting with (1,1) in the bottom left hand corner. The x value is in the horizontal direction.



The **knight** can move in an "L" shape. This equates to either:

- two squares vertically and one square horizontally
- or two squares horizontally and one square vertically

Input Format

The chess piece will be entered first, followed by the x and y location as integers.

Output Format

You should first output the number of moves that are possible on a single line, followed by each x and y location of all the squares that can be reached in a single move on separate lines.

Sample Input

4 4

Sample Output

8
(2, 3)
(2, 5)
(3, 2)
(3, 6)
(5, 2)
(5, 6)
(6, 3)
(6, 5)

```
size = 8


def knight(x, y):
    moves = []
    if x - 2 > 0:
        if y + 1 <= size:
            moves.append((x - 2, y + 1))
        if y - 1 > 0:
            moves.append((x - 2, y - 1))
    if x + 2 <= size:
        if y + 1 <= size:
            moves.append((x + 2, y + 1))
        if y - 1 > 0:
            moves.append((x + 2, y - 1))

    if y - 2 > 0:
        if x + 1 <= size:
            moves.append((x + 1, y - 2))
        if x - 1 > 0:
            moves.append((x - 1, y - 2))
    if y + 2 <= size:
        if x + 1 <= size:
            moves.append((x + 1, y + 2))
        if x - 1 > 0:
            moves.append((x - 1, y + 2))
    return moves


def output(moves):
    moves.sort()
    for move in moves:
        print(move)


s = input().split(" ")
result = knight(int(s[0]), int(s[1]))
print(len(result))
output(result)
```

B5 Problem Statement

Soccer Tournament

In a 4 team round-robin tournament, each team plays all the other teams. There will be six games. We will input the games results from the first team's view as "W" for win, "L" for loss, and "T" for tie. We will list them: A vs B; A vs C; A vs D; B vs C; B vs D; C vs D.

A team's score is determined by 3 points for each win and 1 point for ties. Losses are zero points.

Given the six letters giving the games results, output the teams' names and scores in order from highest to lowest.

Example:

Input: W W W L L T

Output: A 9pts; C 4 pts; D 4 pts; B 0 pts

B5 Solution
Soccer Tournament

```
int main()
{
	char T, Team[4];
	int S, Score[4];
	char Gab, Gac, Gad, Gbc, Gbd, Gcd;
	int j, i;
	Team[0] = 'A'; Team[1] = 'B'; Team[2] = 'C'; Team[3] = 'D';
	Score[0] = 0; Score[1] = 0; Score[2] = 0; Score[3] = 0;
	std::cout << "\nenter game A vs B result: ";
	std::cin >> Gab;
	if (Gab == 'W') { Score[0] = Score[0] + 3; }
	if (Gab == 'L') {Score[1] = Score[1] + 3;}
	if (Gab == 'T') { Score[0]++; Score[1]++; }
	std::cout << "\nenter game A vs C result: ";
	std::cin >> Gac;
	if (Gac == 'W') { Score[0] = Score[0] + 3; }
	if (Gac == 'L') { Score[2] = Score[2] + 3; }
	if (Gac == 'T') { Score[0]++; Score[2]++; }
	std::cout << "\nenter game A vs D result: ";
	std::cin >> Gad;
	if (Gad == 'W') { Score[0] = Score[0] + 3; }
	if (Gad == 'L') { Score[3] = Score[3] + 3; }
	if (Gad == 'T') { Score[0]++; Score[3]++; }
	std::cout << "\nenter game B vs C result: ";
	std::cin >> Gbc;
	if (Gbc == 'W') { Score[1] = Score[1] + 3; }
	if (Gbc == 'L') { Score[2] = Score[2] + 3; }
	if (Gbc == 'T') { Score[1]++; Score[2]++; }
	std::cout << "\nenter game B vs D result: ";
	std::cin >> Gbd;
	if (Gbd == 'W') { Score[1] = Score[1] + 3; }
	if (Gbd == 'L') { Score[3] = Score[3] + 3; }
	if (Gbd == 'T') { Score[1]++; Score[3]++; }
	std::cout << "\nenter game C vs D result: ";
	std::cin >> Gcd;
	if (Gcd == 'W') { Score[2] = Score[2] + 3; }
	if (Gcd == 'L') { Score[3] = Score[3] + 3; }
	if (Gcd == 'T') { Score[2]++; Score[3]++; }
	for(j = 0; j < 3; j++) {
		for(i = 0; i < 3; i++) {
			if (Score[i] < Score[i + 1]) {
				S = Score[i]; T = Team[i]; Score[i] = Score[i + 1];
				Team[i] = Team[i + 1];  Score[i + 1] = S; Team[i + 1] = T;
			}
		}
	}
	std::cout << "\nAnswer: ";
	for(i = 0; i < 4; i++) {
		std::cout << "\n" << Team[i] << " " << Score[i];
	}
	return 0;
}
```

B6 Problem Statement
Numbers Puzzle

Puzzles are fun brain teasers with a wide range of scenarios. For this problem, you are presented with a sequence of *n* whole numbers that are randomly ordered. Somewhere in this sequence of numbers, there will be a 0. This 0 is fixed in place in the sequence and cannot be moved. To solve this puzzle, you must sort the numbers in ascending order without moving the 0. Your algorithm must use the smallest number of moves possible. Every time one number is swapped with another number counts as one move. Your program should output the state of the sequence of numbers after each move is made. After the puzzle is solved, your program should output the number of moves made.

Input Format

The input for this program will be a sequence of *n* integers.

Output Format

The output of the sequence after each move must be on its own line. After the puzzle is solved, the number of moves made should be outputted on its own line.

Sample Input

4 0 1 6 7 2

Sample Output

1,0,4,6,7,2
1,0,2,6,7,4
1,0,2,4,7,6
1,0,2,4,6,7
4

```
numbers = [int(x) for x in input().split(’ ’)]
swap = 0
slot = numbers.index(0)
start = 0
size = len(numbers)
if slot == 0:
    start = 1

while start < size:
    if start != slot:
        m = min(i for i in numbers[start: size] if i > 0)
        i = numbers.index(m, start, size)
        if i > start:
            numbers[slot] = numbers[start]
            numbers[start] = numbers[i]
            numbers[i] = numbers[slot]
            numbers[slot] = 0
            swap += 1
            print(",".join(str(x) for x in numbers))
    start += 1

print(swap)
```